The Event List

Many mathematical modeling paradigms abstract a set of states from the system of interest. A state is a mathematical variable that represents an important aspect of the system. The Discrete Event paradigm for simulation modeling views the evolution of the model in time as a collection of state trajectories, each of which is piecewise constant. Each state trajectory can be thought of as a graph for the corresponding state variable in time. A discrete event simulation model is specified by defining all the states and specifying the rules by which they change values. Piecewise constant state trajectories imply that states change value only at particular instants of time. These instants of state changes are termed events, hence the term "Discrete Event." It may be expedient for the modeler to define events that do not necessarily invoke a state change. However, every time some state does change value necessarily implies that some event has taken place.

The Event List (or "Future Event List") and its manipulation comprise the engine that drives a Discrete Event Simulation model. The manipulation of the Event List is done by the Basic Discrete Event Algorithm, which is responsible for managing the passage of simulated time by firing events in temporal order. For models written in general-purpose languages (C, Java, Visual Basic, Fortran, etc.) the Event List and the timing routine must all be coded by the user. In most simulation languages, the Event List and timing routine are provided by the language itself and run in the background, generally unseen by the user. Some languages are explicitly discrete event oriented: SIMLIB (described in Chapter 2 of Law and Kelton) is one example, and Simkit (which we will use in this course) is another. Other event-oriented simulation languages include SIGMA, GASP and early versions of SIMSCRIPT. For a discrete event language the simulation programmer only has to supply the code for events: state transitions, scheduling other events, and typically some bookkeeping tasks, such as output reports, statistics, etc. No simulated time passes during events; rather, simulated time passes only between events.

The Event List amounts to a "to do" list for the simulated world. At any simulated time epoch it is simply a list of what is scheduled to occur and when. Each item on the list corresponds to an event that contains information about which event is to occur and when it is to occur.

The Basic Discrete Event Algorithm is shown in Figure 1 and proceeds as follows. First the Event List is checked to see if it is empty; an empty Event List implies that nothing further can ever occur, so the simulation ends. If there is at least one event on the Event List, the simulation clock is advanced to the time of that event, which is then removed from the Event List. The state transition for that event is then executed. After the state changes, the event may schedule other events to occur. After these scheduled events (if any) are put on the Event List, the algorithm starts all over with checking to see if the Event List is empty. The key feature is the fact that the Event List stores all events in increasing temporal order, regardless of the time they were originally scheduled. It should be clear that the most natural implementation of an Event List is a priority queue, with the priority being the time of the occurrence of the event notice. In most simulation languages it is possible to have a “stop” event that stops the algorithm and empties the Event List.

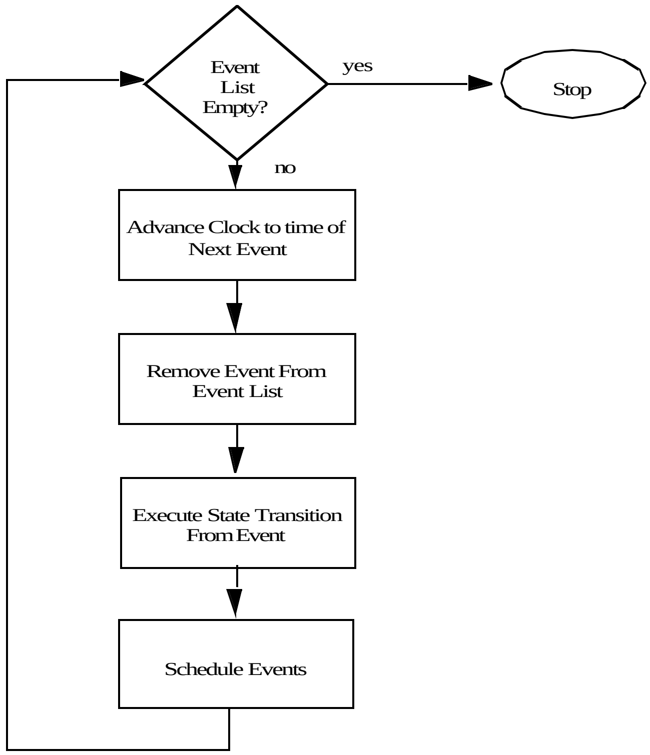

Figure 1. Basic Discrete Event Algorithm

A pseudo-code version of the Basic Discrete Event Algorithm is given in Figure 2.

```
while (Event List is not empty)

       Advance time to next event

       Remove next event from Event List

       Perform state transition for event

       Schedule other events
```

Figure 2. Pseudo-Code Version of Basic Discrete Event Algorithm

The representation of the algorithm in Figure 2 is closer in form to current implementations.

Suppose that the current simulated time is S and that the event in question is to be scheduled in t time units. The corresponding event would thus be scheduled to occur at time S + t. That event would be put on the Event List so that it is ahead of every other event on the list whose scheduled time is greater than S + t and behind every event whose scheduled time is less than S + t. If the event is scheduled to occur at *exactly* the same time as another event, then some kind of tie-breaking rule must be used to order the events. Remember that the Event List processes the events one at a time.

For example, consider a discrete event model of a G/G/3 queue at time 12.943 with the Event List as shown in Figure 3. The current event (that has just completed) is "`StartService`," which is an event corresponding to the onset of the service of a customer. The state transition for an `StartService` event is to decrement the number of customers waiting in the queue (`numberInQueue`) by one and to also decrement the number of available servers (`numberAvailableServers`) by one as well. This effect of this state transition is to change the value of the state variable `numberInQueue` from 11 to 10 and the value of the state variable `numberAvailableServers` from 1 to 0. This transition is indicated above the first Event List in Figure 3.

. In this particular case, a service time of 2.834 was generated, so an `EndService` was scheduled to occur at time (12.943 + 2.834) = 15.777, as reflected in the first Event List in Figure 3.[1]

Now control passes to the Event List manager. Since the Event List is not empty, time is advanced to the next scheduled event, an `Arrival` event at time 13.403 in this case. The state transition for the `Arrival` event is executed and the next event(s) scheduled. The state transition for the `Arrival` event changes the value of `numberArrivals` from 25 to 26 and `numberInQueue` from 10 to 11. The `Arrival` event then schedules the next`Arrival` event. An interarrival time of 1.395 is generated, so the next `Arrival` event is scheduled to occur at (13.403 + 1.395) = 14.338, which is reflected in the second Event List. Note how the `Arrival` event is inserted into the list according to its time as compared with the other scheduled events. Thus, it will not be the next event to occur; rather, the next event at this point will be the previously scheduled `EndService` event.

```
numberInQueue: 11 => 10
numberAvailableServers: 1 => 0
Time: 12.943    Current Event: StartService     [14]
 ** Event List --  **
13.403  Arrival
13.491  EndService
14.564  EndService
15.777  EndService
 ** End  of Event List --  **

numberArrivals: 24 => 25
numberInQueue: 10 => 11
Time: 13.403    Current Event: Arrival  [25]
 ** Event List --  **
13.491  EndService
14.338  Arrival
14.564  EndService
15.777  EndService
 ** End  of Event List --  **
```

Figure 3. Two Event List Snapshots for a G/G/3 Queue[2]

1. Note that the state transitions for each event are shown *above* the Event List snapshot and that the Event List reflects the scheduling that has taken place by the execution of the event.
2. For the terminally curious, the numbers in brackets refer to the event counts: by time 13.403 the StartService event has occured 14 times and there have been 25 Arrival events.

Exercises

1. 1. For the Event List in show the subsequent snapshots of the Event List when remaining interarrival times are {2.135, 3.422} and successive service times are {1.512, 4.441}. Is there any additional information you need to keep track of?

2. 2. A G / G /2 queue has the following interarrival times and service times (in minutes): Arrival times: {4.1, 6.2, 0.8, 9.6, 12.8, 19.8} Service times: {29.2, 4.2, 16.2, 36.3, 7.6}. The servers are initially idle. Perform a hand simulation using these interarrival and service times, showing the status of the state space and the event list after every event in your simulation. Stop your simulation after the fifth customer has completed service.

References

[1] Banks, J., J. Carson, and B. Nelson. 1996. Discrete-Event System Simulation, Second Edition. Prentice-Hall.

[2] Law, A. and D. Kelton. 2000. Simulation Modeling and Analysis, Third Edition, McGraw Hill.

[3] Fujimoto, R. 2000. Parallel and Distributed Simulation Systems. John Wiley.